Lab1: GPUs: Fall 2015

Name: Subhankari Mishra

1. C version of the problem is included attached in programs folder. The file name is “sequential.c”. The program can be run for different data sizes by changing the value of the variable width for data size and iterations for the number of iterations.

2. Cuda version of the programs are also included in the attached folder with file names as “cuda_version_<data_size>.cu”.

3. Below are the optimizations that were made to the sequential program:

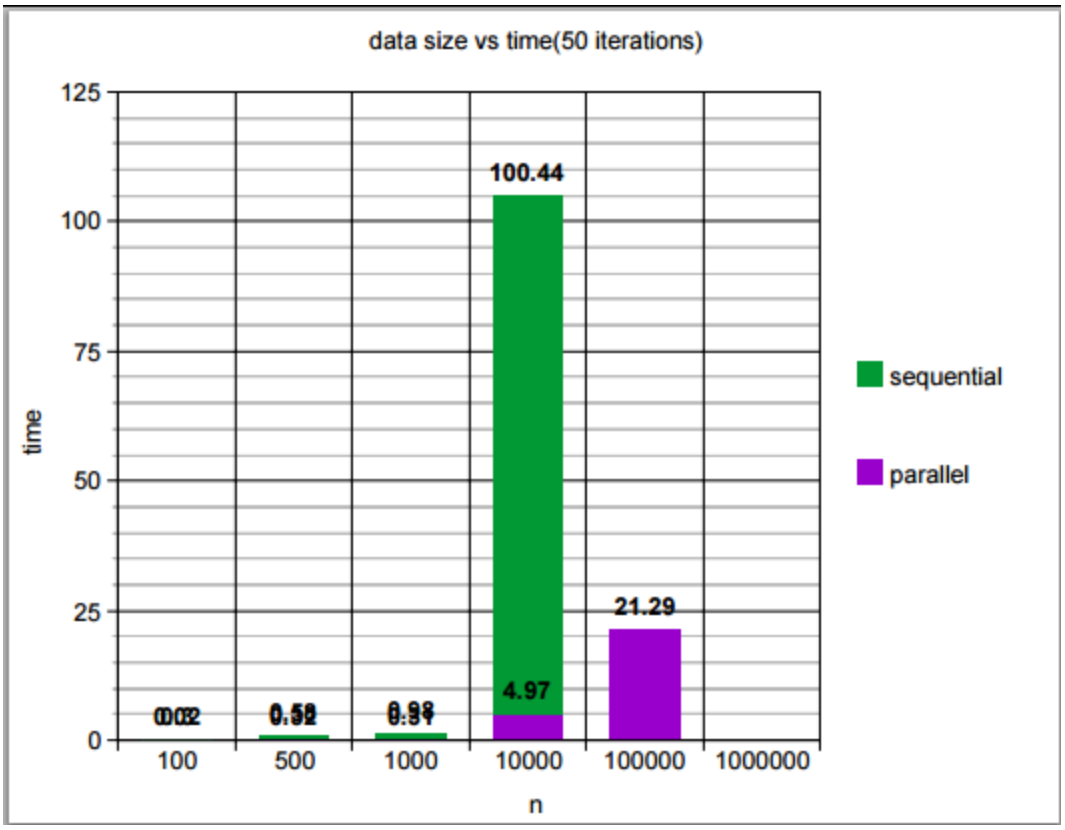
    a. The matrix initialization is made using the GPU kernel. Please refer file cuda_version_100.cu.
    b. The calculation of temperature for the inner points is also made using the GPU kernel. Please refer any of the attached files with .cu extension.
    c. The blocks/grid and threads/block values is set depending on the data size to accommodate max number of threads as per the configuration of Titan Black.
    d. No kernel configuration could accommodate the data size of n = 100,000 and n = 1000,000.
    e. I also implemented tiling for n = 10000. Please refer cuda_version_10000.
    f. For n = 100,000, it was observed that the iterations only calculate the values from the edge to a width equal to the number of iterations, rest of the cells remain zero.
    So the program can be optimized only to store the values required for the calculation and allocating the same to the GPU kernel and not storing the values which always remain zero.
    Please refer cuda_version_100000.cu. This program has bugs could not fix it due to lack of time although it runs through all the threads but gives incorrect result. Probably erroneous index calculation due to changed data dimension.
    g. Another approach which is experimented with is slicing the data into segments which can be processed at a time by the GPU kernel without memory overflow.
    Executing each segment each time than storing the data on main program then again triggering the GPU kernel for the next segment. Basically processing the data in fragments. Please refer cuda_version_2_100000.cu. This program also has bugs and gives segmentation fault.
    h. For n = 1000,000, I could not allocate the data, the memory overflows.

4. The programs were executed for n = 100; 500; 1,000; 10,000; 100,000 and 1,000,000.
For n = 100,000 and 1,000,000 the sequential code ran out of memory. It could not allocate the memory to the array. For rest of the data sizes the program ran successfully.
For n = 1,000,000, the cuda-version ran out of memory in the main program itself. For all other data sizes the programs ran successfully when correct kernel configuration were given based on the configuration of Titan Black device and applying the above mentioned optimizations.

5. Below are the average running time in seconds for the programs over 50 iterations:

Data Size (n)	Avg Time taken by sequential code(in secs)	Avg time taken by Cuda version(in secs)
100	0.020	0.30
500	0.58	0.32
1000	0.98	0.31
10000	100.44	4.97
100000	Ran out of memory	21.29
1000000	Ran out of memory	Ran out of memory

6. Below graph represents the time taken by the sequential program and cuda-version for the given data sizes and the number of iterations.



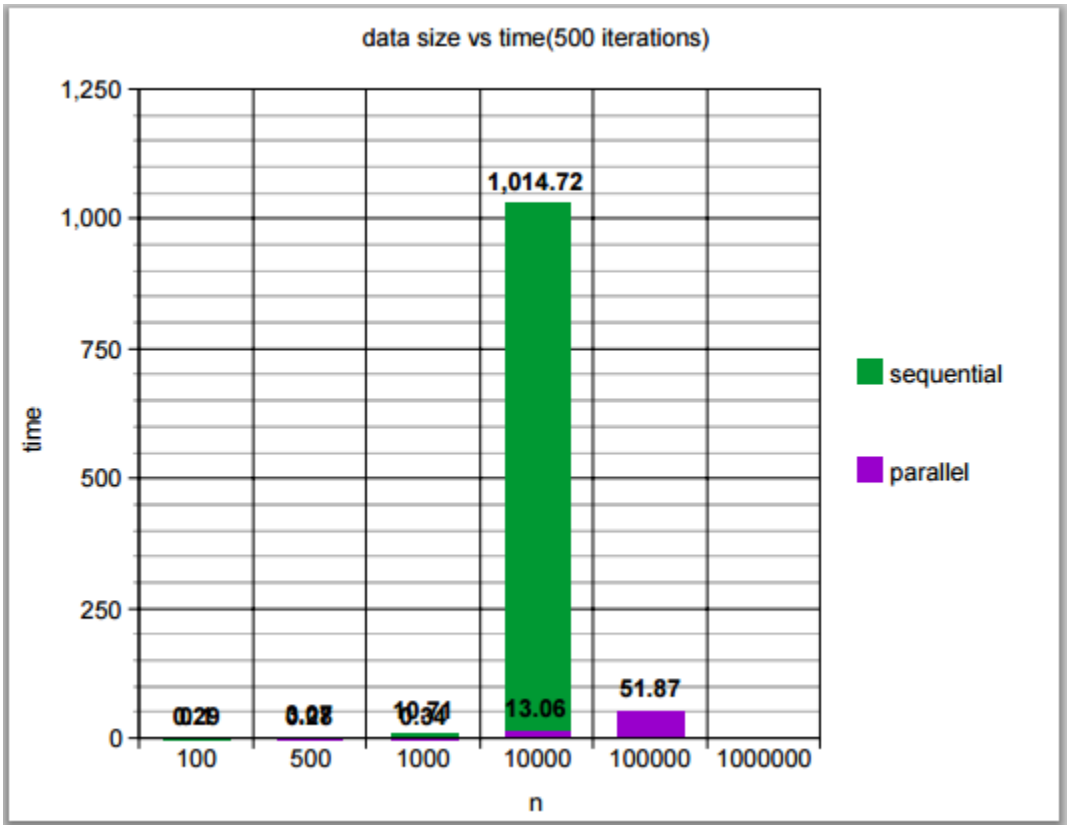
7. Conclusions:
    a. As per the data experimented GPU is beneficial at higher data size like n = 10000 and 100000 as the speedup is very high as compared to the sequential code.
    b. The speedup is at its lowest at n = 100. As the data size is very small the difference between the running time of the sequential code and parallel code is very small. Here, the speedup gained by the GPU is being compensated by the overhead of data transfer for the small amount of data to GPU and them from GPU to host.

c. The speedup is at its highest for n= 10000 and n = 100000, due to massive parallelism which makes the overhead of data transfer negligible.

8. Below are the average running time in seconds for the programs over 50 iterations:

Data Size (n)	Avg Time taken by sequential code(in secs)	Avg time taken by Cuda version(in secs)
100	0.10	0.29
500	3.07	0.28
1000	10.71	0.34
10000	1014.72	13.06
100000	Ran out of memory	51.87
1000000	Ran out of memory	Ran out of memory

Below graph represents the time taken by the sequential program and cuda-version for the given data sizes and the number of iterations.



Conclusions:

- The GPU is more beneficial for large data size which can make the complete utilization of the parallel threads in GPU. As per the given data, the GPU is most beneficial at n = 10000 and 100000 due to massive parallellism.

- The speedup is at its lowest for n = 100 due same reasons as discussed in conclusion section of 50 iterations.
- The speed up is at its highest for n = 10000 and n = 100000 due the same reason as discussed above.

9. Upon increasing the number of iterations the execution time of the programs increased for both sequential and cuda-version, but the amount of increase in the cuda-version is negligible whereas for sequential execution the execution time increases proportional to the increase in the number of iterations. As in our case the sequential execution time increased by approx. 10 times for each data size but the increase in execution time in cuda version for small data size is in negligible and for large data size approx. 2-3 times.